MARTE based design flow for Partially Reconfigurable Systems-on-Chips

Imran Rafiq Quadri, Alexis Muller, Samy Meftali, Jean-Luc Dekeyser

MARTE based design flow for Partially Reconfigurable Systems-on-Chips

Imran Rafiq Quadri*, Alexis Muller*, Samy Meftali* and Jean-Luc Dekeyser*
*INRIA Lille Nord Europe - LIFL - USTL - CNRS, 40 avenue Halley, 59650 Villeneuve d'Ascq, FRANCE
Email:{Firstname.Lastname}@inria.fr

I. Introduction

Systems-on-Chip (SoCs) are considered an integral solution for designing embedded systems, for targeting complex intensive parallel computation applications. As advances in SoC technology permit integration of increasing number of hardware resources on a single chip, the targeted application domains such as software-defined radio are become increasingly sophisticated. The fallout of this complexity is that the system design, particularly software design, does not evolve at the same pace as that of hardware leading to a significant *productivity gap*. *Adaptivity* and *reconfigurability* are also critical issues for SoCs which must be able to cope with end user environment and requirements.

An effective solution to SoC Co-design problem consists in raising the design abstraction levels. The important requirement is to find efficient design methodologies that raise the design abstraction levels to reduce overall SoC complexity. A suitable control model is also required for managing the system adaptivity; it should be generic enough to be applied to both software and hardware design aspects. While several control models exist, automata based control [1] are promising as they incorporate aspects of modularity present in component based approaches for describing SoC in an incremental fashion to build these complex systems.

Once a suitable control model is chosen, implementation of these adaptive SoC systems can be carried out via FPGAs. These FPGAs are inherently reconfigurable in nature. State of the art FPGAs can change their functionality at *runtime*, known as Partial Dynamic Reconfiguration (PDR) [2]. These FPGAs also support internal self dynamic reconfiguration, in which an internal controller (a *hardcore*/*softcore* embedded processor) manages the reconfiguration aspects.

Finally the usage of *concurrent high level design approach* in development of real-time embedded systems is also increasing to address the compatibility issues related to SoC Co-design. High abstraction level SoC co-modeling design approaches have been developed in this context, such as Model-Driven Engineering (MDE) [3] that specify the system using the UML graphical language. MDE enables high level system modeling (of both software and hardware) with the possibility of integrating heterogeneous components into the system. Usage of UML for system description increases the system comprehensibility. This allows designers to provide high-level descriptions of the system that easily illustrate the internal concepts (task/data parallelism, data dependencies and hierarchy). These specifications can be reused, modified or extended due to their graphical nature. Finally *Model transformations* [4] can be carried out to generate executable models or source code from high level models.

Gaspard [5] is an MDE-based SoC co-design framework that uses the UML MARTE profile [6] to model real-time and embedded systems; and allows to move from high level MARTE specifications to different execution platforms such as RTL synthesis in FPGAs [7]. It exploits the inherent *parallelism* included in repetitive constructions of hardware elements or regular constructions such as application loops. The applications targeted by Gaspard also focus on a specific application domain, that of data-parallel applications.

In this paper we present a generic control semantics for expressing partial reconfigurability in SoCs. The introduced control semantics are introduced in the MARTE standard and subsequently in Gaspard; and are specified at an high abstraction level. Integration of this control allows to focus on FPGA synthesis and is specially oriented towards PDR. The goal is to specify part of the reconfigurable system at a high abstraction level: notably the reconfigurable region and the reconfiguration controller in a dynamically reconfigurable FPGA. Afterwards, using model transformations, the gap between high level specifications and low implementation details can be bridged to automatically generate the code required for the creation of bitstream(s) for final FPGA implementation. Finally we present a case study illustrating the modeling and final implementation of a dynamically reconfigurable key integral part of an anti-collision radar detection system. This part is based on delay estimation using a correlation algorithm. This part is modeled at an high abstraction level using the MARTE profile in the Gaspard framework. Afterwards using the model transformations present in our design flow, we have generated the necessary RTL level code for synthesis on a target FPGA using commercial tools. It should be observed that the final code generation from the high level models usually is carried out in a few seconds resulting in a huge save in the overall system conception development time.

The rest of this paper is organized as follows. Related works are detailed in section II. An overview of the MDE-based Gaspard framework is provided in section III. Section IV describes the control model in Gaspard, while Section V presents our case study. Finally section VIII gives the conclusion of the paper.

II. Related Works

There are several works that use high abstraction level methodologies for defining embedded systems and SoCs. MoPCoM [8] targets modeling and code generation of embedded systems using SysML block diagrams. In [9], a SynDEx based design flow is presented to manage SoC reconfigurability via implementation in FPGAs, with the application and architecture parts modeled as components. Similarly in [10], a UML profile is described along with a tool set for modeling, verification and simulation of real-time embedded systems. Reconfigurability in SoC can be related to available system resources such as available memory, computation capacity and power consumption. An example of a component based approach with adaptation mechanisms is provided in [11]; e.g. for switching between resources [12].

In [13],[14], the authors concentrate on verification of real-time embedded systems in which the control is specified at a high abstraction level via UML state machines and collaborations; by using model checking. However, control methodologies vary in nature as they can be expressed via different forms such as Petri Nets [15], or other formalisms such as mode automata [1].

Mode automata extend synchronous dataflow languages with an imperative style, but without many modifications of language style and structure [1]. They are mainly composed of *modes* and *transitions*. In an automaton, each mode has the same interface. Equations are specified in modes. Transitions are associated with conditions, which serve to act as triggers. Mode automata can be composed together in parallel. They enable formal validation by using the synchronous technology. Among existing UML based approaches allowing for design verification are the Omega project [10] and Diplodocus [16]. These approaches utilize model checking and theorem proving.

In domain of SoC adaptivity and especially partial dynamic reconfiguration in FPGAs, Xilinx initially proposed two design flows, which were not very effective leading to new alternatives. An effective modular approach for 2-D shaped reconfigurable modules was presented in [17]. [18] implemented modular reconfiguration using a horizontal slice based bus macro in order to connect the static and partial regions. They then placed arbitrary 2-dimensional rectangular shaped modules using routing primitives [19]. This approach has been further refined in [20]. In 2006, Xilinx introduced the *Early Access Partial Reconfiguration Design Flow* [21] that integrated concepts of [17] and [18]. Works such as [22],[23] focus on implementing softcore internal configuration ports on Xilinx FPGAs such as Spartan-3, that do not have the hardware Internal Configuration Access Port (ICAP) reconfigurable core, for implementing PDR. Works such as [24] and [25] illustrate usage of customized ICAPs. Finally in [26], the ICAP reconfigurable core is connected with Networks-on-chip (NoC) implemented on dynamically reconfigurable FPGAs.

In comparison to the above related works, our proposition takes into account the following domains: SoC co-design, control/data flow, MDE, UML MARTE profile, SoC reconfigurability and PDR for FPGAs; which is the novelty of our design framework.

III. Gaspard: a SoC Co-design Framework

Gaspard [5] is a MARTE [6] compliant SoC co-design framework that enables to model *software applications*, *hardware architectures* and their *allocations* in a concurrent manner. Once models of software applications and hardware architectures are defined, the functional parts (such as application tasks and data) can be mapped onto hardware resources (such as processors and memories) via *allocation(s)*. Gaspard also introduces a *deployment* level that links hardware/software components with intellectual properties (IPs). This section will be elaborated in Section IV.

For the purpose of automatic code generation from high level models, Gaspard adopts MDE model transformations towards different execution platforms, such as targeted towards FPGA synthesis [7], as shown in Figure.1. Model transformation chains allow moving from high abstraction levels to low enriched levels. Usually, the initial high level models contain only domain-specific concepts, while technological concepts are introduced seamlessly in the intermediate levels.

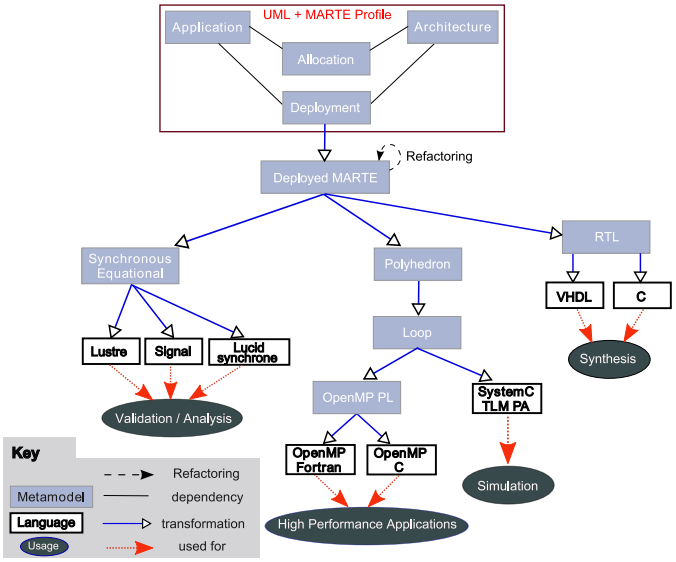

Figure.1: A global view of the Gaspard framework

IV. A Reactive Control Model

We first describe the generic control semantics which can be integrated into the different levels (application, architecture and allocation) in SoC co-design. Several basic control concepts, such as `Mode Switch Component` and `State Graphs` are presented first. Then a basic composition of these concepts, which builds the mode automata, is discussed. This modeling derives from the mode conception in mode automata. The notion of exclusion among modes helps to separate different computations. As a result, programs are well structured and fault risk is reduced. We then use the Gaspard SoC co-design framework for utilization of these concepts.

A. Modes

A mode is a distinct method of operation that produces different results depending upon the user inputs. A mode switch component in Gaspard contains at least more than one mode; and offers a switch functionality that chooses execution of one mode, among several alternative present modes [27]. The mode switch component in Figure.2 illustrates such a component having a *window* with multiple tabs and interfaces. For instance, it has an *m* (mode value input) port as well as several i_d (data input) and o_d (data output) input and output ports. The switch between the different modes is carried out according to the mode value received through *m*.

The modes, M_1, ..., M_n, in the mode switch component are identified by the mode values: m_1, ..., m_n. Each mode can be hierarchical or elementary in nature; and transforms the input data i_d into the output data o_d. All modes have the same interface (i.e. i_d and o_d ports). The activation of a mode relies on the reception of mode value m_k by the mode switch component through *m*. For any received mode value m_k, the mode runs exclusively. It should be noted that only mode value ports, i.e., *m*; are compulsory for creation of a mode switch component, as shown in Figure.2. Thus other type of ports are represented with dashed lines.

B. State graphs

A state graph in Gaspard is similar to state charts [28], which are used to model the system behavior using a state-based approach. It can be expressed as a graphical representation of transition functions as discussed in [29]. A state graph is composed of a set of vertices, which are called *states*. A state connects with other states through directed edges. These edges are called *transitions*. Transitions can be conditioned by some events or Boolean expressions. A special label *all*, on a transition outgoing from state *s*, indicates any other events that do not satisfy the conditions on other outgoing transitions from *s*. Each state is associated with some mode value specifications that provide mode values for the state. A state graph in Gaspard is associated with a `Gaspard State Graph` as shown in Figure.2.

C. Combining modes and state graphs

Once mode switch components and state graphs are introduced, a `MACRO` component can be used to compose them together. The macro in Figure.2 illustrates one possible composition. In the macro, the Gaspard state graph produces a mode value (or a set of mode values) and sends it (them) to the mode switch component. The latter switches the modes accordingly. Some data dependencies (or connections) between these components are not always necessary, for example, data dependency between I_d and i_d. They are drawn with dashed lines in Figure.2. The illustrated figure is used as a basic composition, however, other compositions are also possible, for instance, one Gaspard state graph can control several mode switch components.

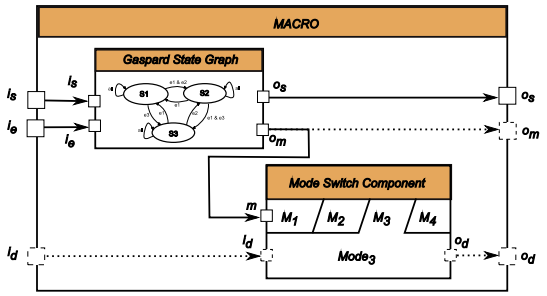

Figure.2: An abstract representation of a macro structure

V. Control at Deployment level

In this section we explain control integration at another abstraction level in SoC co-design. This level deals with linking the modeled application and architecture components to their respective IPs. We thus elaborate on the component model of this *deployment* level in the particular case of the Gaspard SoC co-design framework.

A. Deployment in Gaspard

The Gaspard deployment level enables one to precise a specific IP for each elementary component of application or architecture, among several possibilities [30]. The reason is that in SoC design, a functionality can be implemented in different ways. For example, an application functionality can either be optimized for a processor, thus written in C/C++, or implemented as a hardware accelerator using Hardware Description Languages (HDLs). Hence the deployment level allows the designer to differentiate between the hardware and software functionalities; and allows moving from platform-independent high level models to platform-dependent models for eventual implementation.

A `VirtualIP` expresses the functionality of an elementary component, independently from the compilation target. For an elementary component *K*, it associates *K* with all its possible IPs. The desired IP(s) is (are) then selected by the SoC designer by linking it (them) to *K* via an `implements` dependency. Finally, the `CodeFile` (not illustrated in the chapter) determines the physical path related to the source/binary code of an IP, along with required compilation options.

B. Multi-Configuration approach

Currently in deployment level, an elementary component can be associated with only one IP among the different available choices (if any). Thus the result of the application/architecture (or the mapping of the two forming the overall system) is a static one. This collective composition is termed as a *Configuration*. The current model transformations for RTL level only allow to generate one hardware accelerator (hence one configuration) for final FPGA effectuation.

Integrating control in deployment allows to create several configurations for the final effectuation(s) in an FPGA. Each configuration is viewed as a collection of different IPs, with each IP associated with its respective elementary component.

We thus link the different modeled IPs with a modeling concept called `Configuration` that links a particular configuration with its respective associated IPs. This Configuration concept is enriched with several attributes such as a `ConfigurationID`, that assigns a unique integer value to each configuration; and the `InitialState` attribute which is a boolean value that determines the initial configuration.

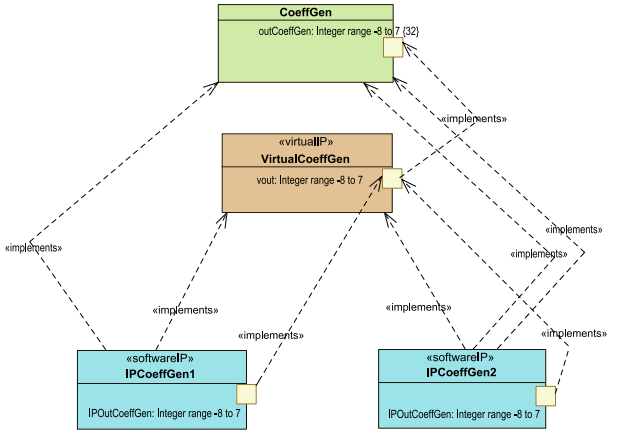

Figure.3: Deployment of an elementary component

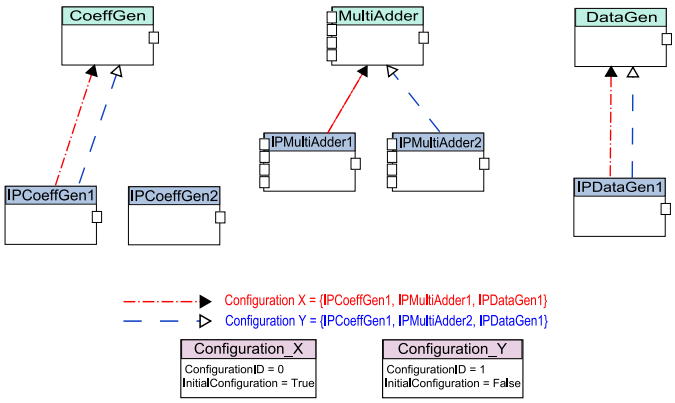

Figure.4: Key dynamically reconfigurable elementary components of our FIR filter application

An elementary component can also be associated with the same IP in different configurations. This point is very relevant to the semantics of partial bitstreams (FPGA configuration files for partial dynamic reconfiguration) which support *glitchless dynamic reconfiguration*. If a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation. If the same IP for an elementary component is present in several configurations, that IP is not changed during reconfiguration. It is thus possible to link several IPs with a corresponding elementary component; and each link specifies a unique configuration. We apply a condition that for any *n* number of configurations with each having *m* elementary components, each elementary component of a configuration must have *at least* one IP. This allows successful creation of a complete configuration for eventual final implementation. This information is then passed onto the control concept modeled in the second phase of deployment level using the model to model transformations.

Figure.3 represents the deployment for one of the elementary components `CoeffGen` present in our case study application which will be elaborated later on. Finally Figure.4 represents an abstract global overview of the deployment semantics related to three elementary components of our application along with the required configurations. A change in implementation of any of these elementary components produces a different final result related to different QoS criteria such as FPGA resources etc.

By modifying the model transformations related to the RTL level, it is possible to generate different hardware accelerators, hence different configurations. Once the configurations are created, each is treated as a source for a partial bitstream. Each partial bitstream signifies a unique implementation, related to a reconfigurable hardware accelerator which is connected to an embedded controller. While this extension allows to create different configurations, the state machine part of the controller is created manually. For automatic generation of this functionality, the deployment extensions are not sufficient. We then use the existing control concepts presented in Section V to solve these issues.

C. Control Model

We first present some concepts which are used in the modeling of mode automata [31]. In Gaspard, data are manipulated in the form of multidimensional arrays. Both data parallelism and task parallelism can be expressed easily via MARTE profile. A `repetitive component` expresses the data parallelism in an application: in the form of sets of input/output `patterns` consumed and produced by the repetitions of the interior `part`. A `hierarchical` component contains several `parts`. Specifically, task parallelism can be described using a hierarchical component in our framework. A `tiler` connector describes the tiling of produced and consumed arrays and thus defines the shape of a data pattern. The `interrepetition dependency` is used to specify an acyclic dependency among the repetitions of the same component. The interrepetition dependency specification leads to the sequential execution of repetitions. A `defaultlink` provides a default value for repetitions linked with an interrepetition dependency, with the condition that the source of dependency is absent.

An interrepetition dependency allows the construction of mode automata from Gaspard control model, which requires two subsequent steps. First, the internal structure of `Repetitive Component` is presented by the `Deployed MACRO` component illustrated in Figure 5. The Gaspard state graph in the macro acts as a state-based controller and the mode switch component achieves the mode switch function. A macro structure represents only a single transition between states. In order to execute continuous transitions as present in automata, the macro should be repeated to have multiple transitions. An interrepetition dependency forces the continuous

sequential execution. This allows the construction of mode automata which can be then executed. Detailed information related to the control model at the deployment level can be found in [31].

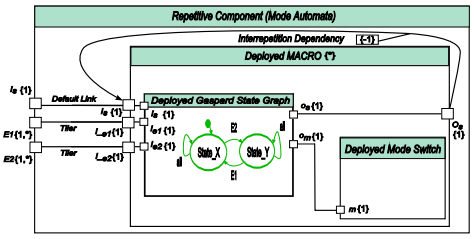

Figure.5: The Deployment Level Mode Automata

VI. Case Study: Partially reconfigurable Delay estimation correlation module

Correlation algorithms are among the type of digital processing largely employed in DSP (digital signal processing) based systems. They offer a large applicability range such as linear phase and stability. A correlation algorithm normally takes some input data values and computes an output which is then multiplied by a set of coefficients. Afterwards the result of this multiplication is added together to produce the final output. While a software implementation can be utilized for implementing this functionality, the correlation functionality will be sequentially executed. Where as a hardware implementation allows the correlation functions to be executed in a parallel manner and thus increases the processing speed.

We propose to study a case where our radar uses a PRBS (Pseudorandom binary sequence) of length of 127 chips. To produce a computation result, the algorithm requires 127 multiplications between 127 elements of the reference code and the last 127 received samples. The result of this multiplication produces 64 data elements. The sum of these 64 data elements produces the final result. This result can be sent as input to other parts of the anti-collision radar detection system.

By integrating a mode automata control model at the deployment level [31], we are able to change the IPs related to key elementary components in order to produce several configurations that are treated as partial bitstreams related to our case study. Switching the configurations and the associated IPs is related to different QoS criteria such as consumed FPGA resources and power consumption levels. We are also able to automatically generate the code which can be implemented on a hardcore/softcore embedded processor for managing the dynamic reconfiguration.

Figure.6 shows the global overview of our design flow. The figure only shows the top hierarchical level of our correlation application. Once the application has been completely modeled using the UML MARTE profile, the model transformations allow to move from high level models to lower detailed models which add detailed information and concepts related to RTL semantics. Finally automatic code generation related to our application and the reconfiguration controller is carried out.

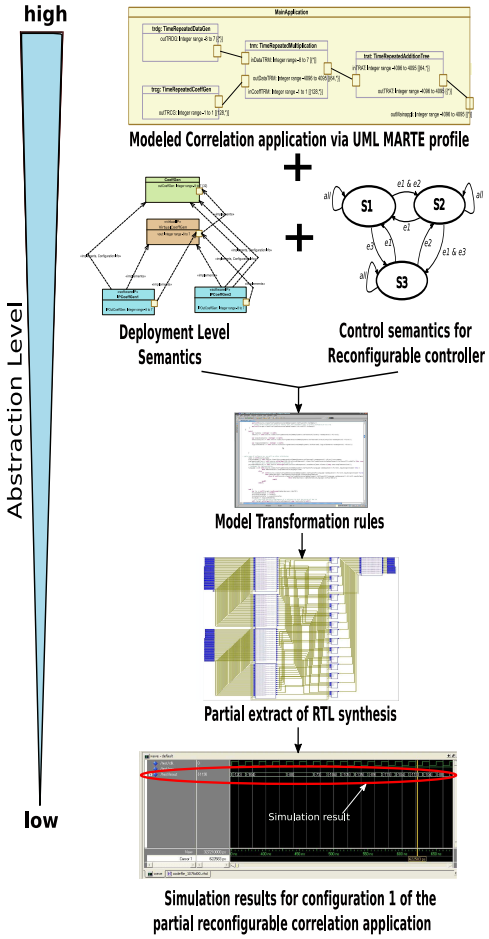

Figure.6: An overviw of our complete design flow

The application contains temporal as well as spatial dimensions which can be easily expressed in our design flow. Similarly, *task parallelism* and *data parallelism* can be specified at the high abstraction levels, and the generated code expresses the parallelism specified at the modeling level. The partially reconfigurable system has been implemented on a Xilinx XC2VP30 Virtex-II Pro FPGA with a PowerPC 405 as a reconfiguration controller with a frequency of 100 MHz. We implemented three configurations on the targeted architecture, two with different IPs related to an multiplication elementary component and a blank configuration. Figure.7 shows the implementation of the first configuration with the multiplication IP written as a DSP functionality. The complete results are shown in Figure.8. The important point to remember is that this methodology can be implemented on any Xilinx FPGA supporting partial dynamic reconfiguration [2].

In conclusion, we have presented a novel design methodology to model complex intensive data-parallel applications. The modeling is carried out using the UML graphical language and the MARTE standard. Afterwards, automatic code generation can be carried out via MDE tools and technologies. Finally the code can be synthesized and implemented on a target FPGA.

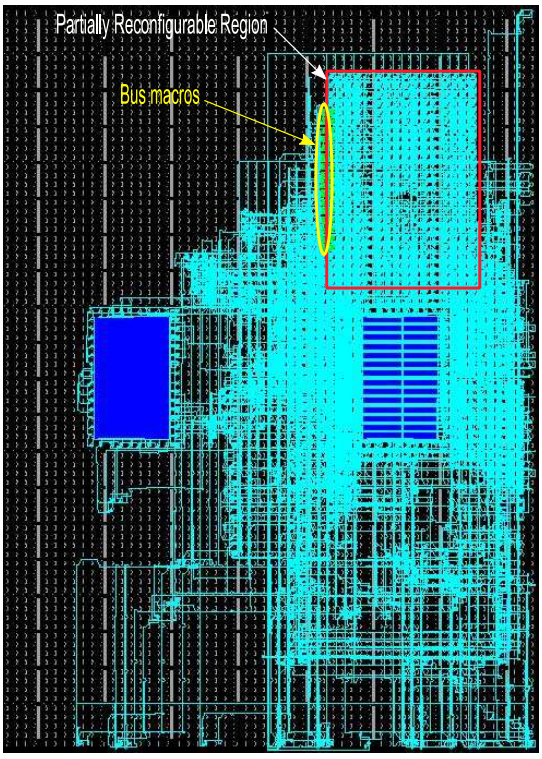

Figure.7: Configuration 1 on an Virtex II-Pro XC2VP30

	FPGA resources			Dynamic Power (mW)	Mean Reconfiguration Time (msecs)
	Slices	Slice Flip Flops	LUTs		
Configuration 1 : Multiplication IP written with DSP functionality	1268/13696 (9.25%)	1995/27392 (7.283%)	1786/27392 (6.520%)	221.42	47.42
Configuration 2 : Multiplication IP written with switch case functionality	1216/13696 (8.878%)	1845/27392 (6.735%)	1748/27392 (6.381%)	214.56	45.38

Figure.8: An overview of the obtained results

REFERENCES

[1] F. Maraninchi and Y. Rémond, "Mode-automata: About modes and states for reactive systems," in *European Symposium On Programming*. Lisbon (Portugal): Springer verlag, Mar. 1998.

[2] P. Lysaght and B. Blodget and J. Mason, "Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *FPL'06*, 2006.

[3] OMG, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.

[4] T. Mens and P. Van Gorp, "A taxonomy of model transformation," in *Proceedings of the International Workshop on Graph and Model Transformation, GraMoT 2005*, 2006, pp. 125–142.

[5] DaRT team, "GASPARD Framework," 2009, http://www.gaspard2.org/.

[6] OMG, "Modeling and analysis of real-time and embedded systems (MARTE)," http://www.omgmarte.org/.

[7] I.-R. Quadri, S. Meftali, and J.-L. Dekeyser, "A model driven design flow for fpgas supporting partial reconfiguration," *International Journal of Reconfigurable Computing*, 2009, Hindawi Publishing Corporation, Tentative publication date : June 2009.

[8] A. Koudri et al, "Using MARTE in the MOPCOM SoC/SoPC Co-Methodology," in *MARTE Workshop at DATE'08*, 2008.

[9] F. Berthelot and F. Nouvel and D. Houzet, "A Flexible system level design methodology targeting run-time reconfigurable FPGAs," *EURASIP Journal of Embedded Systems*, vol. 8, no. 3, pp. 1–18, 2008.

[10] S. Graf, "Omega – Correct Development of Real Time Embedded Systems," *SoSyM, int. Journal on Software & Systems Modelling*, vol. 7, no. 2, pp. 127–130, 2008.

[11] M. Segarra and F. André, "A framework for dynamic adaptation in wireless environments," in *Proceedings of 33rd International Conference on Technology of Object-Oriented Languages (TOOLS 33)*, 2000, pp. 336–347.

[12] J. Buisson, F. André, and J.-L. Pazat, "A Framework for Dynamic Adaptation of Parallel Components," in *ParCo 2005*, 2005.

[13] Latella, D. and Majzik, I. and Massink, M, "Automatic Verification of a Behavioral Subset of UML Statechart Diagrams Using the SPIN Model-Checker," in *Formal Aspects Computing*, vol. 11, 199, pp. 637–664.

[14] T. Schäfer, A. Knapp, and S. Merz, "Model checking UML state machines and collaborations," in *CAV Workshop on Software Model Checking*, ser. ENTCS 55(3), 2001.

[15] B. Nascimento et al, "A partial reconfigurable architecture for controllers based on Petri nets," in *SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design*. ACM, 2004, pp. 16–21.

[16] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet, "A UML-based environment for system design space exploration," *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pp. 1272–1275, Dec. 2006.

[17] P. Sedcole and B. Blodget and J. Anderson and P. Lysaght and T. Becker, "Modular Partial Reconfiguration in Virtex FPGAs," in *International Conference on Field Programmable Logic and Applications, FPL'05*, 2005, pp. 211–216.

[18] J. Becker and M. Huebner and M. Ullmann, "Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations," in *VLSI'03*, 2003.

[19] M. Huebner and C. Schuck and M. Kiihnle and J. Becker, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-Time Adaptive Microelectronic Circuits," in *ISVLSI'06*, 2006.

[20] C. Schuck and M. Kuhnle and M. Hubner and J. Becker, "A framework for dynamic 2D placement on FPGAs," in *IPDPS 2008*, 2008.

[21] Xilinx, "Early Access Partial Reconfigurable Flow," 2006, http://www.xilinx.com/support/prealounge/protected/index.htm.

[22] Bayar, S., and Yurdakul, A, "Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP)," in *2nd HiPEAC workshop on Reconfigurable Computing, HiPEAC 08*, 2008.

[23] K. Paulsson and M. Hubner and G. Auer and M. Dreschmann and L. Chen and J. Becker, "Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGA," *International Conference on Field Programmable Logic and Applications, FPL 2007*, pp. 351–356, 2007.

[24] C. Claus and F.H. Muller and J. Zeppenfeld and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *IPDPS 2007*, pp. 1–7, 2007.

[25] A. Cuoccio and P. R. Grassi and V. Rana and M. D. Santambrogio and D. Sciuto, "A Generation Flow for Self-Reconfiguration Controllers Customization," *Forth IEEE International Symposium on Electronic Design, Test and Applications, DELTA 2008*, pp. 279–284, 2008.

[26] R. Koch and T. Pionteck and C. Albrecht and E. Maehle, "An adaptive system-on-chip for network applications," in *IPDPS 2006*, 2006.

[27] O. Labbani and J.-L Dekeyser and Pierre Boulet and É. Rutten, "Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach," in *Proceedings of the International Workshop MARTES: Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[28] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987, article note found. [Online]. Available: citeseer.ist.psu.edu/harel87statecharts.html

[29] A. Gamatié, E. Rutten, and H. Yu, "A Model for the Mixed-Design of Data-Intensive and Control-Oriented Embedded Systems," INRIA, http://hal.inria.fr/inria-00293909/fr, Research Report RR-6589, July 2008.

[30] R. B. Atitallah, E. Piel, S. Niar, P. Marquet, and J.-L. Dekeyser, "Multilevel MPSoC simulation using an MDE approach," in *SoCC 2007*, 2007.

[31] J.-L. Dekeyser, A. Gamatié, S. Meftali, and I.-R. Quadri, "Models for Co-Design of Hetero-geneous Dynamic Multiprocessor SoCs," in *Book Chapter in Heterogeneous Embedded Systems: Design Theory and Practice*. Springer verlag, 2010.